Report on results of Trend Analysis experiment.

28 September 2002

Norman F. Schneidewind, Fellow, IEEE
Naval Postgraduate School
Monterey, CA 93943, USA
Voice: (831) 656-2719
Fax : (831) 372-0445
nschneid@nps.navy.mil

Summary

This is Report # 2 in a series of reports on the NASA IV&V Facility Project "Investigation of the Risk to Software Reliability and Maintainability of Requirements Changes". This report covers the trend analysis experiment. In Report # 1, we developed an approach for identifying requirements change risk factors as predictors of reliability and maintainability problems. Our case example consisted of twenty-four Space Shuttle change requests, nineteen risk factors, and the associated failures and software metrics of the Space Shuttle "Three Engine Out" software (designated "OIO" in this report). The approach can be generalized to other NASA domains with numerical results that would vary according to the application.

In Report # 1, we identified four Space Shuttle requirements change risk factors that had a statistically significant effect on reliability. These were the following: the amount of memory space required to implement a requirements change ("space"), the number of requirements issues ("issues"), the number of modifications ("mods"), and the size of the change ("sloc"), in that priority order. In this report, we address the following three types of trends:

1. Trends of reliability and risk factors: evaluate cumulative failures as a function of cumulative risk factors and vice versa.

2. Trend and shape metrics: evaluate product reliability and maintainability and the stability of the process that produces the product [SCH99].

3. Trends in fault correction: evaluate trends in the numbers of corrected and remaining faults and trends in fault correction times [SCH012].

The purpose of these evaluations is to provide a comprehensive view of trends in reliability risk (i.e., risk of faults and failures induced by changes in requirements) that may be incurred by deficiencies in the process (e.g., lack of precision in requirements). With this comprehensive analysis in hand, the software manager would have valuable information for deciding whether the software is safe to deploy. Although we use examples from the Space Shuttle, the approach is general and could be applied to any NASA application.

This report is organized as follows: 1. Introduction, 2. Objectives, 3. Analysis of Results, 4. Conclusions, 5. Technology Transfer, 6. Future Research, References, and Bibliography.

1. Introduction

1.1 Trends of Reliability and Risk Factors

Having identified the leading risk factors in Report # 1, in this report, we use these risk factors and their associated failure data to develop actual (i.e., empirical) trend and prediction equation plots of reliability versus risk factors and vice versa. We use cumulative risk factors and cumulative failures for this purpose. The objective of developing the trend equations and plots is to predict reliability as a function of risk factors and vice versa for future Operational Increments (OIs) of the Space Shuttle. In addition, we examine the rate of change of these functions in order to identify the sensitivity of reliability to changes in requirements change requests. The details of applying risk factor reliability predictors are covered in section 3.1.

1.2 Trend and Shape Metrics

The trends mentioned in section 1.1 are useful but only address a single release of the software (e.g., OIO). To address the interaction between reliability and maintainability with process stability across and within releases, trend and shape metrics are used. By chronologically ordering metric values by release date, we obtain discrete functions in time that can be analyzed for trends across releases. In addition to trends in metrics, the shapes of metric functions provide indicators of process stability. We use the trends of these metrics across releases to analyze long-term process stability. We use the shapes of these metrics within a release to analyze short-term process stability. The rationale of these metrics is it is desirable to show reliability growth (i.e., trend metrics) and that it is better to reach important points in the growth of reliability sooner than later (i.e., shape metrics). If we reach these points late in testing, it is indicative of a process that is late in achieving stability. In sections 3.2 and 3.3, we apply trend and shape metrics, respectively, to make a comparison of the reliability risk and process stability of release OIO with other releases of the same software system. Thus, the software manager is able to identify product and process anomalies among releases and to prioritize the allocation of resources to product and process improvement according to the results of the comparison.

1.3 Trends in Fault Correction

In general, software reliability models have focused on modeling and predicting failure occurrence and have not given equal priority to modeling the fault correction process. However, there is a need for fault correction prediction, because there are important applications that fault correction modeling and prediction support. These are the following: predicting whether reliability goals have been achieved, developing stopping rules for testing, formulating test strategies, and rationally allocating test resources. This trend analysis provides a more accurate evaluation of the reliability risk of deploying the software than would be the case if we only considered the occurrence of failures. In the latter case, the reliability of the software would be under stated because reliability predictions would not account for the removal of faults. The details of applying trends in fault correction are covered in section 3.4.

2. Objectives

Identify the attributes of requirements that cause the software to be unreliable and quantify the relationship between requirements risk and reliability. If these attributes can be identified, then policies can be recommended to NASA for recognizing these risks and avoiding or mitigating them during development. Extend and validate our work in this area on the Space Shuttle [SCH011] to the Goddard Space Flight Center and the Jet Propulsion Laboratory software projects.

3. Analysis of Results

3.1 Trends of Reliability and Risk Factors

We identify thresholds of risk factors (i.e., the attributes of a requirements change that can induce reliability risk) for predicting when the number of failures would become excessive (i.e., rise rapidly with the risk factor) [SCH011].

Two of the most important requirements risk factors of the Space Shuttle, as measured by their negative effect on software reliability, are *space* and *issues*. The former is defined as the amount of memory space required to implement the requirement change and the latter is defined as the number of possible conflicts among requirements. In [SCH011], it was determined that these two risk factors had the highest statistically significant relationship with reliability (i.e., the greater the cumulative memory space required to implement changes and the greater the number of cumulative conflicting requirements issues caused by the changes, the greater the negative effect on reliability).

An example is shown in Figure 1, where cumulative failures are plotted against cumulative memory space for both actual and predicted data. The later was formed using non-linear regression, where a second-degree polynomial was the best fit. The prediction equation, which is shown on the figure, has a residual standard error of 1.53266 on 4 degrees of freedom. The figure shows that when memory space reaches 2688 words, actual cumulative failures reach three and climb rapidly thereafter. Another example is shown in Figure 2, where cumulative failures are plotted against cumulative requirements issues, for both actual and predicted cases. The prediction equation, which is shown on the figure, has a residual standard error of 1.07889 on 5 degrees of freedom. When *issues* reach 272, actual cumulative failures reach 3 and climb rapidly thereafter. In both cases, a cumulative failure count of 3 has been identified as a *critical* value. Although the counts of 2688 words and 272 issues provide estimates of the threshold to use in controlling the reliability of the next version of the software, the next version may not exhibit bends in the curves at the same value of risk factor. Therefore, the prediction equations and plots are an attempt to generalize the relationship between risk factors and reliability, such that they can be used to predict cumulative failures for *any* given value of cumulative risk factor. This process would be repeated across versions with the prediction equations being updated as more data is gathered. Thresholds would be identified for several risk factors. This would provide multiple alerts of low reliability.

Additional insight about the relationship between risk factors and reliability can be gained by plotting the first derivative of the prediction equations in Figures 1 and 2 (i.e., rate of change). These are shown in Figures 3 and 4 for *space* and *issues*, respectively. Because the equation in Figure 1 is a second-degree polynomial, its derivative in Figure 3 is linear; thus, *space* has a predicted linear growth rate. In contrast, because the equation in Figure 2 is an exponential, its derivative is also an exponential and is simply the original function multiplied by a constant. This plot is shown in Figure 4. In comparing Figures 3 and 4, the implication is that we should have more concern about the negative effect on reliability of *issues* because of its predicted exponential growth rate.

Finally, for the next version of this software, we want to predict the cumulative values of risk factors that correspond to given values of cumulative failures, particularly critical values. Figures 5 and 6, show the plots corresponding to the equations on the figures. These equations and plots were obtained by solving the equations of Figures 1 and 2 for cumulative risk factor as a function of cumulative failures. For example, if cumulative failures equal to 3 are considered critical, this would correspond to 1596 words of memory (Figure 5) and an issue count of 232 (Figure 6).

3.2 Trend Metrics

Although looking for a trend on a graph is useful, it is not a precise way of measuring and comparing trends, particularly if the graph has peaks and valleys and the measurements are made at discrete points in time. In analyzing the relationship between product reliability and process stability, we solved this problem by developing a generalized relative Change Metric (CM) that represents trend information (e.g., changes in reliability across releases) in a single metric [SCH98]. CM is independent of the scales of the measured quantities. We will use this metric to measure changes in reliability across releases of the software and compare them to see whether trends are favorable, indicating reliability growth and process stability, or unfavorable, indicating lack of reliability growth and process instability. The following is an example of computing CM for the reliability metric failures/KLOC:

1. Note the change in a metric from one release to the next (i.e., release j to release j+1).

2.a. If the change is in the desirable direction (e.g., Failures/KLOC decrease), treat the change in 1 as positive.

b. If the change is in the undesirable direction (e.g., Failures/KLOC increase), treat the change in 1 as negative.

3. a. If the change in 1 is an increase, divide it by the value of the metric in release j+1.
b. If the change in 1 is a decrease, divide it by the value of the metric in release j.

4. Compute the average of the values obtained in 3, taking into account sign. This is the change metric (CM). The CM is a quantity in the range –1, 1. A positive value indicates a favorable trend; a negative value indicates an unfavorable trend. The numeric value of CM indicates the degree of stability or instability. The standard deviation of these values can also be computed. The average of the CM for a set of metrics can be computed to obtain an overall

change metric. An example of calculating CM for Mean Time to Failure (MTTF) and Failures/KLOC is shown in Table 1 for various Operational Increments (i.e., releases) of Space Shuttle software. Figures 7 and 8 show the corresponding plots of Mean Time to Failure and Failures/KLOC, respectively, across the releases. The results suggest slight process instability for MTTF and slight stability for Failures\KLOC. Overall, we would not conclude that the process in unstable.

We note in Figure 7 that OIO – the operational increment analyzed for reliability risk in section 3.1 -- has a relatively low MTTF (unfavorable) and in Figure 8 a relatively low Failures/KLOC (favorable). In addition, in Table 1, the relative changes from OIJ to OIO are negative – unfavorable. Overall, these results suggest that OIO is a candidate for additional inspections and testing, in particular to increase its MTTF.

Table 1: Example Computations of Change Metric (CM)

Operational Increment	MTTF (Days)	Relative Change	Failures/KLOC	Relative Change
A	179.7		0.750	
B	409.6	0.562	0.877	-0.145
C	406.0	-0.007	1.695	-0.483
D	192.3	-0.527	0.984	0.419
E	374.6	0.487	0.568	0.423
J	73.6	-0.805	0.238	0.581
O	68.8	-0.068	0.330	-0.272
	CM	**-0.060**	**CM**	**0.087**

3.3 Shape Metrics

In addition to trends in metrics, the shapes of metric functions provide indicators of maintenance stability. We use shape metrics to analyze the stability of an individual release and the trend of these metrics across releases to analyze long-term stability. The rationale of these metrics is that it is better to reach important points in the growth of product reliability sooner than later. If we reach these points late in testing, it is indicative of a process that is late in achieving stability. We use the following types of shape metrics:

1. Direction and magnitude of the slope of a metric function (e.g., failure rate decreases asymptotically with test time). Using failure rate as an example within a release, it is desirable that it rapidly decrease towards zero with increasing test time and that it have small values.

2. Percent of test time at which a metric function changes from unstable (e.g., increasing failure rate) to stable (e.g., decreasing failure rate) and remains stable. Across releases, it is desirable that the test time at which a metric function becomes stable gets progressively smaller.

3. Percent of test time at which a metric function increases at a maximum rate in a favorable direction (e.g., failure rate has maximum negative rate of change). Using failure rate as an

example, it is desirable for it to achieve maximum rate of decrease as soon as possible, as a function of test time.

4. Test time at which a metric function reaches its maximum value (e.g., test time at which failure rate reaches its maximum value). Using failure rate as an example, it is desirable for it to reach its maximum value (i.e., transition from unstable to stable) as soon as possible, as a function of test time.

5. Risk: Probability of not meeting reliability and safety goals (e.g., time to next failure should exceed mission duration), using various shape metrics as indicators of risk. Risk would be low if the conditions in 1-4 above obtain.

3.3.1 Failure Rate

We now apply the principles of section 3.3. to the failure rate function of OIO. In the short-term (i.e., within a release), we want the Failure Rate (1/MTTF) of an OI to decrease over an OI's Test Time, indicating increasing reliability. Practically, we would look for a decreasing trend, after an initial period of instability (i.e., increasing rate as personnel learn how to maintain new software). In addition, we use various shape metrics, as defined previously, to see how quickly we can achieve reliability growth with respect to test time expended. Furthermore, Failure Rate is an indicator of the risk involved in using the maintained software (i.e., an increasing failure rate indicates an increasing probability of failure with increasing use of the software).

Failure Rate = Cumulative Number of Failures During Test/ Cumulative Test Time (1).

We plot Equation (1) for OIO, using actual failure data, in Figure 9 against Test Time since the release of OIO. Figure 9 does show that short-term stability is achieved (i.e., failure rate asymptotically approaches zero with increasing Test Time). In addition, this curve shows when the failure rate transitions from unstable (positive Failure Rate) to stable (negative Failure Rate). The figure also shows how risk is reduced with decreasing Failure Rate as the maintenance process stabilizes. We use this plot to assess whether we have achieved short-term stability in the maintenance process (i.e., whether Failure Rate decreases asymptotically with increasing Test Time). If we obtain contrary results, this would be an alert to investigate whether this is caused by: 1) greater functionality and complexity of the OI as it is being maintained, 2) a maintenance process that needs to be improved, or 3) a combination of these causes.

We use equation (2) [ANS93, SCH99] to compute the predicted Failure Rate, where i is a vector of time intervals for $i \geq s-1$, and s is the starting time interval for using failures counts for computing parameters α and β, in equation (2) [SCH93].

$$f(i) = \alpha(EXP(-\beta(i-s+1))) \quad (2).$$

A 30-day interval has been found to be convenient as a unit of Space Shuttle test time because testing can last for many months or even years. Thus, this is the unit used in Figure 9, where we apply the Shape Metrics described in section 3.3 and show the following events in

intervals, where the predictions were made at 12.0 intervals and Total Test Time is 37.0 intervals from the release of the OI by the contractor to NASA:

Release time: 0 interval

Launch time: 13.27 intervals

Predicted time of maximum Failure Rate: 6.0 intervals

Actual time of maximum Failure Rate: 12.0 intervals

Predicted Maximum Failure Rate: .4215 failures per interval
This occurs at 16.22 % of Total Test Time (6.0 intervals)

Actual Maximum Failure Rate: .4167 failures per interval
This occurs at 32.43 % of Total Test Time (25.0 intervals)

Predicted maximum rate of change of Failure Rate: -.0060 failures per interval per interval
This occurs at 16.22 % of Total Test Time (6.0 intervals)

Actual maximum rate of change of Failure Rate: -.0136 failures per interval per interval
This occurs at 67.57 % of Total Test Time (37.0 intervals)

In Figure 9, stability is achieved after the maximum failure rate occurs. This is at $i = s - 1$ (i.e. $i = 6$ intervals) for predictions because equation (2) assumes a monotonically decreasing failure rate, whereas the actual failure rate increases, reaches a maximum at 12.0 intervals, and then decreases. Once stability is achieved, risk decreases. We also note in Figure 9 that the maximum rate of change of the actual Failure Rate occurs at 25.0 intervals. This is a key point in achieving reliability growth of the product and maturity of the process because at this time reliability is improving at the maximum rate. Of course, the Space Shuttle would not be launched until stability has been achieved, as shown in Figure 9.

Although launch took place at 13.27 intervals, the above predictions and actual data represent a *retrospective* analysis of events. The discrepancy between predicted and actual results is accounted for by the fact that whereas the actual data were produced by accounting for events over the Total Test Time of 37.0 intervals, predictions were made after only 12.0 test intervals had elapsed. In practice, the actual Failure Rate curve would be plotted as failures occur in order to determine when stability has been reached (i.e., Failure Rate turns from increasing to decreasing). In addition, predictions of failure rate would be made for the important reason that at any time in testing, future failure events would be unknown.

3.4 Trends in Fault Correction

The final analysis involves trends in fault correction counts and corrections times, along with stopping rules that are based on modeling and predicting the fault correction process. We continue the analysis of OIO -- the Space Shuttle OI that was at risk in the early phases of its development -- as the analysis in sections 3.1, 3.2, and 3.3 indicated.

3.4.1 Remaining Faults

The predicted number of remaining faults, after the correction process has been operative for time T, is given by equation (3):

$$N(T) = D(T) - C(T) \quad (3),$$

where D (T) is the number of failures detected and C (T) is the number of faults corrected by time T.

This equation is based on the assumption that all the faults that exist in the software have been predicted by D (T) [SCH012]. A more conservative prediction is obtained by predicting the detected failures over the life of the software D (T_L), as in equation (4) [SCH97], and then using equation (5) as the predicted remaining faults.

$$D(T_L) = \alpha/\beta + X_{s-1} \quad (4),$$

where X_{s-1} is the observed failure count in the range 1, s-1.

$$N(T_L) = D(T_L) - C(T) \quad (5).$$

The number of remaining faults N (T), equation (3), plotted as a function of test time, as illustrated in Figure 10 for Shuttle OIJ and OIO, can be used as a stopping rule for testing. We used an upper probability limit of .90, meaning that the probability is .90 that N (T) is less than or equal to its ordinate values for a given test time, or .10 that these values are exceeded. The practical significance of this plot is that it is highly unlikely that reliability could be improved by testing for more than 30 intervals. This figure is interesting because it shows a cross over between the OIs at T = 11.5. This would imply that OIJ should be given higher priority before T = 11.5 and lower priority after it. By priority, we mean the order of testing and allocation of personnel and computer resources. We consider N (T) useful because it can be used with the critical value of remaining faults R_c -- a reliability threshold. For example, for R_c = 1 in Figure 10, testing would be terminated for OIJ at T = 13.5 and for OIO at T = 26.

3.4.2 Reliability Improvement

In the previous section, we used absolute quantities (e.g., number of remaining faults) for developing test strategies and assessing reliability. In this section, we use the relative quantity p (T), the proportion of faults remaining at time T, which is related to r (T), the proportion of faults corrected at time T, by equation (6):

$$p(T) = 1 - r(T) = 1 - C(T)/D(T) \quad (6).$$

A plot of equation (6) is shown in Figure 11 for OIJ and OIO. We do this because two software systems could have experienced different numbers of failures but have equal numbers of remaining faults. In this case, the software with fewer failures would have achieved greater progress in reliability improvement, as measured by p (T). However, the use of a threshold seems more intuitive when applied to N (T). In practice, both measures could be used.

3.4.3 Test Scheduling

We can anticipate test requirements and do proactive test scheduling by using the predicted amount of test time required to correct a given number of faults [SCH012] . In addition, a plot of multiple software systems shows how the systems compare in test requirements. An example is shown in Figure 12 for OID and OIO, where the predictions are for the case of zero faults corrected at the time of prediction, and a .90 upper probability limit. Because the values for OIJ are virtually identical to those for OID, the former is not plotted. We see that the difference in test time between the OIs increases with increasing number of faults. This result implies that relatively large quantities of resources -- personnel and computer time -- would be required to correct the faults in OIO, and that this need accelerates as the number of faults to correct increases.

3.4.4 Validation

An example of a validation, using the test – failure – fault correction scenario for OIO is shown in Table 2. This involved determining from the collected data when failures occurred and when faults were corrected. The actual delay between failure occurrence and fault correction was estimated by examining, manually, the Shuttle Discrepancy Reports (i.e., reports that document deviations between specified and observed software behavior) to determine the disposition of the fault (i.e., the release and release date on which the fault was corrected). This was a laborious process because although failures are recorded in electronic files, there is no electronic file of fault corrections with correction dates. We had to infer the correction dates from the release dates.

Either the event column of Table 2 shows when a failure occurred or a fault was corrected. In some cases multiple failures or corrected faults occurred in the same interval; these occurrences are signified by the plural form in the event column. The next column shows the test time T when the events occurred followed by the actual values of cumulative number of failures detected D (T), cumulative number of faults corrected C (T), and number of remaining faults N (T), the difference between D (T) and C (T). The next section shows the predictions for D (T), C (T), and N (T). Notice at the top of Table 2 the statement about the range of prediction, which is $T > s-1$. For OIO, $s = 9$; therefore, the predictions start at interval $T = 9.07$

The last section of Table 2 shows the squares of the differences between actual and predicted values for computing the Mean Square Error (MSE) at the bottom of the table. We computed MSE rather than Mean Relative Error because the latter would have required division by zero in those cases where $C(T) = 0$. We consider the MSE values for OIO to be sufficiently low to validate the predictions.

Table 2: OIO (Predictions for T > s-1 = 8)										
		Actual Values			Predictions			Squared Error		
Event	T	D(T)	C(T)	N(T)	D(T)	C(T)	N(T)	D(T)	C(T)	N(T)
Failure	5.77	1	0	1						
Failure	5.90	2	0	2						
Failure	7.53	3	0	3						
Correction	9.07	3	1	2	3.20	1.06	2.14	0.04	0.00	0.02
Failures	11.47	5	1	4	3.62	1.49	2.13	1.90	0.24	3.50
Corrections	16.80	5	4	1	4.49	3.37	1.11	0.26	0.39	0.01
Failure	24.67	6	4	2	5.60	4.50	1.10	0.16	0.25	0.82
Corrections	29.40	6	6	0	6.18	6.09	0.09	0.03	0.01	0.01
Failure	36.17	7	6	1	6.92	6.84	0.08	0.01	0.71	0.86
Correction	45.77	7	7	0	7.79	7.73	0.06	0.63	0.54	0.00
				Mean Square Error				**0.43**	**0.31**	**0.75**

4. Conclusions

Based on the results of the *retrospective* analysis that has been presented, using OIO as an *example*, we conclude that it is feasible to perform a variety of trend analysis that can be used for the following purposes:

a. Having identified the statistically significant risk factors, as was demonstrated with discriminant analysis in Report 1 for OIO, we used these risk factors as predictors of reliability. We developed prediction equations and identified thresholds of cumulative risk factors and cumulative failures that would allow these quantities to be predicted as a function of the other quantity. When the predictions exceed the thresholds or critical values, they serve as alerts to software management to give priority attention to the software releases that appear to be at risk.

b. We applied trend metrics across releases to compare OIO's reliability and maintainability with other OIs of the Space Shuttle; in addition, this analysis provided an assessment of the long-term stability of the process that produced these OIs. We applied shape metrics to OIO to analyze its empirical and predicted failure rates and the short-term stability of the process that produced it. With these results in hand, the software manager is able to identify product and process anomalies among releases and to prioritize the allocation of resources to product and process improvement according to the results of the comparison.

c. Finally, understanding that the software reliability predictions are limited in accuracy when the effect of fault correction is not considered, we included trends in fault correction counts and fault correction times. This trend analysis provided a more accurate evaluation of the reliability risk of deploying OIO than would be the case if we had only considered the occurrence of failures. In the latter case, the reliability of OIO would be under stated because reliability predictions would not account for the removal of faults. We performed a validation on OIO with respect to predicted failure detection, fault correction, and remaining faults, and obtained good correspondence with the actual values.

5. Technology Transfer

To transfer this technology to the NASA Centers, a possible recommendation is that any requirements change to mission critical software -- either new requirements or changes to existing requirements -- would be subjected to a quantitative risk analysis. The payoff to these organizations would be to reduce the risk of mission critical software *not* meeting its reliability goals during operation. In addition to stating that a risk analysis would be performed, the policy would specify the risk factors to be analyzed (e.g., number of modifications of a requirement or *mod level*) and their threshold or critical values. Once high-risk areas of the software have identified, they would be subjected to detailed tracking throughout the development process. For example, on the Space Shuttle, rigorous inspections of requirements, design documentation, and code have contributed more to achieving high reliability than any other process factor. Thus, it would be prudent to consider adapting this process technology to other NASA projects because the potential payoff in increased reliability would be significant. The objective of these policy changes is to prevent the propagation of high-risk requirements through the various phases of software development.

Our research results can be generalized to any mission critical system. The Space Shuttle data was used because it was available. The risk factors that were used are applicable to any NASA software project. The benefits to other programs of applying the research results would be significant because these programs would be able to determine at an early stage whether proposed requirements changes, either new or changed requirements, are likely to put their systems at risk in terms of unacceptable reliability.

6. Future Research

Future research will involve applying the methodology to another OI of the Space Shuttle and identifying the statistically significant risk factors and thresholds to see whether they match the ones identified in this research. If this were the case, the other part of the methodology – trends analysis – would be applied to this later OI to see whether the analysis would be applicable to this software.

References

[ANS93] Recommended Practice for Software Reliability, R-013-1992, American National Standards Institute/American Institute of Aeronautics and Astronautics, 370 L'Enfant Promenade, SW, Washington, DC 20024, 1993.

[SCH93] Norman F. Schneidewind, "Software Reliability Model with Optimal Selection of Failure Data", IEEE Transactions on Software Engineering, Vol. 19, No. 11, November 1993, pp. 1095-1104.

[SCH97] Norman F. Schneidewind, "Reliability Modeling for Safety Critical Software", IEEE Transactions on Reliability, Vol. 46, No.1, March 1997, pp.88-98.

[SCH98] Norman F. Schneidewind, “How to Evaluate Legacy System Maintenance”, *IEEE Software*, Vol. 15, No. 4, July/August 1998, pp. 34-42. Also translated into Japanese and reprinted in: Nikkei Computer Books, Nikkei Business Publications, Inc., 2-1-1 Hirakawacho, Chiyoda-Ku, Tokyo 102 Japan, 1998, pp. 232-240.

[SCH99] Norman F. Schneidewind, "Measuring and Evaluating Maintenance Process Using Reliability, Risk, and Test Metrics", IEEE Transactions on Software Engineering, Vol. 25, No. 6, November/December 1999, pp. 768-781.

[SCH011] Norman F. Schneidewind, "Investigation of the Risk to Software Reliability and Maintainability of Requirements Changes", Proceedings of the International Conference on Software Maintenance, Florence, Italy, 7-9 November 2001, pp. 127-136.

[SCH012] Norman F. Schneidewind; “Modelling the Fault Correction Process”, Proceedings of The Twelfth International Symposium on Software Reliability Engineering, Hong Kong, 27-30 November, 2001, pp. 185-190.

Bibliography

C. Billings, et al, "Journey to a Mature Software Process", IBM Systems Journal, Vol. 33, No. 1, 1994, pp. 46-61.

Siddhartha R. Dalal and Allen A. McIntosh, "When to Stop Testing for Large Software Systems with Changing Code", IEEE Transactions on Software Engineering, Vol. 20, No. 4, April 1994, pp. 318-323.

S. R. Dalal and C. L. Mallows, “When Should One Stop Testing Software”, Journal of the American Statistical Association, Vol. 83, No. 403, 1988, pp. 872-879.

Willa Ehrlich, Bala Prasanna, John Stampfel, and Jar Wu, "Determining the Cost of a Stop-Test Decision", IEEE Software, March 1993, pp. 33-42.

Swapna S. Gokhale, Teebu Phillip, and Peter N. Marinos, “A Non-Homogeneous Markov Software Reliability Model with Imperfect Repair”, Proceedings of the International Performance and Dependability Symposium, Urbana-Champaign, IL, 1996, 10 pages.

Swapna S. Gokhale, Peter N. Marinos, Michael R. Lyu, and Kishor S. Trivedi, “Effect of Repair Policies on Software Reliability”, Proceedings of Computer Assurance, Gaithersburg, MD, 1997, 10 pages.

Chin-Yu Huang, Sy-Yen Kuo, Michael R. Lyu, and Jung-Hua Lo, “Quantitative Software Reliability Modeling from Testing to Operation”, Proceedings of the Eleventh International Symposium on Software Reliability Engineering, IEEE Computer Society Press, Los Alamitos, CA, October 8-10, 2000, pp. 72-82.

Daniel R. Jeski, “Estimating the Failure Rate of Evolving Software Systems”, Proceedings of the Eleventh International Symposium on Software Reliability Engineering, IEEE Computer Society Press, Los Alamitos, CA, October 8-10, 2000, pp. 52-61.

Ted Keller, Norman F. Schneidewind, and Patti A. Thornton "Predictions for Increasing Confidence in the Reliability of the Space Shuttle Flight Software", Proceedings of the AIAA Computing in Aerospace 10, San Antonio, TX, March 28, 1995, pp. 1-8.

Ted Keller and Norman F. Schneidewind, “Successful Application of Software Reliability Engineering for the NASA Shuttle”, Software Reliability Engineering Case Studies, International Symposium on Software Reliability Engineering, November 3, Albuquerque, New Mexico, November 4, 1997, pp. 71-82.

Michael R. Lyu (Editor-in-Chief), Handbook of Software Reliability Engineering, Computer Society Press, Los Alamitos, CA and McGraw-Hill, New York, NY, 1995, pp. 95-98.

John D. Musa, et al, Software Reliability: Measurement, Prediction, Application, McGraw-Hill, New York, 1987.

Allen P. Nikora, Norman F. Schneidewind, and John C. Munson, IV&V Issues in Achieving High Reliability and Safety in Critical Control Software, Final Report, Jet Propulsion Laboratory, National Aeronautics and Space Administration, Pasadena, California, January 19, 1998.

Nozer D. Singpurwalla, "Determining an Optimal Time Interval for Testing and Debugging Software", IEEE Transactions on Software Engineering, Vol. 17, No. 4, April 1991, pp. 313-319.

Carol Smidts, “A Stochastic Model of Human Errors in Software Development: Impact of Repair Times”, Proceedings of the Tenth International Symposium on Software Reliability Engineering, IEEE Computer Society Press, Los Alamitos, CA, November 1-4, 1999, pp 94-103.

Min Xie and M. Zhao, “The Schneidewind Software Reliability Model Revisited”, Proceedings of the Third International Symposium on Software Reliability Engineering, IEEE Computer Society Press, Los Alamitos, CA, Research Triangle Park, NC, October 7-10, 1992, pp. 184-192.

Figure 1: Failures vs. Memory Space

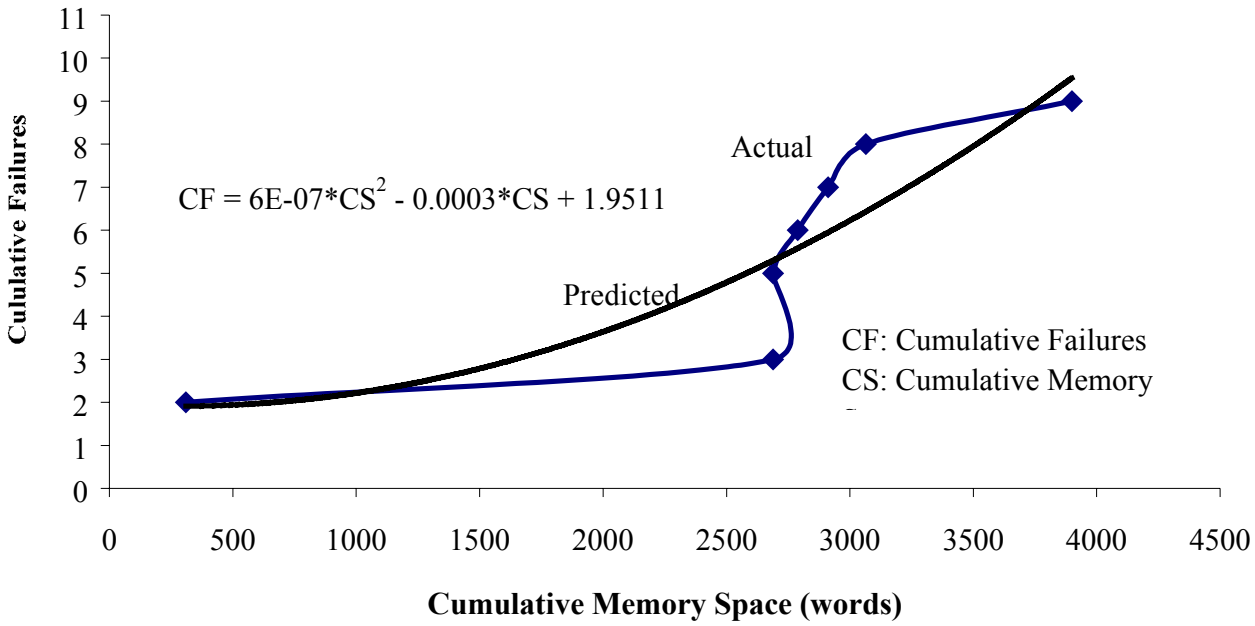

Figure 2: Failures vs. Issues

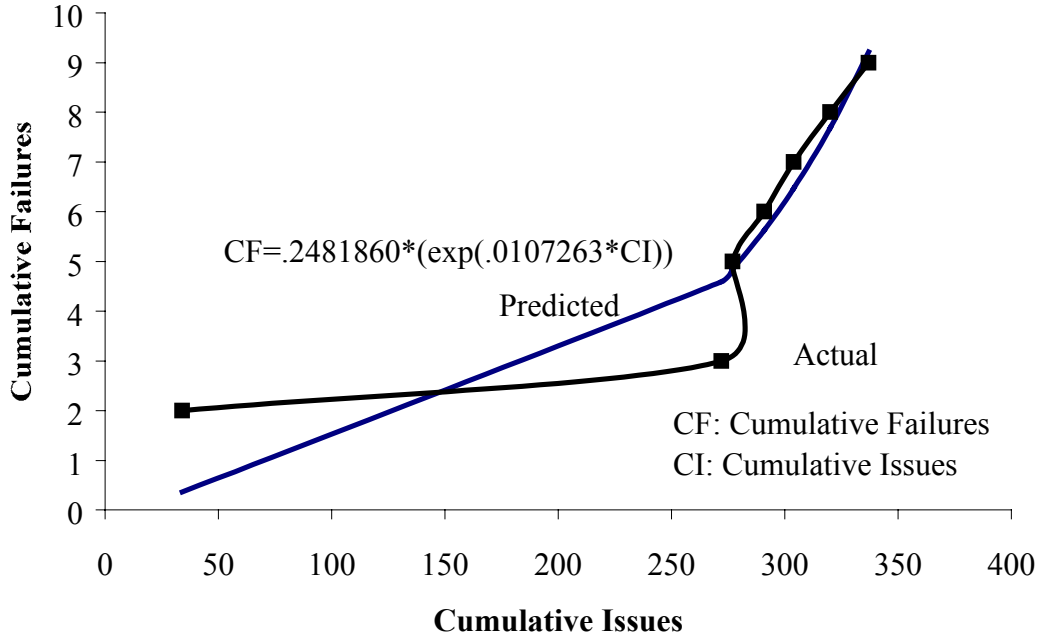

Figure 3. Rate of Change of Failures with Memory Space

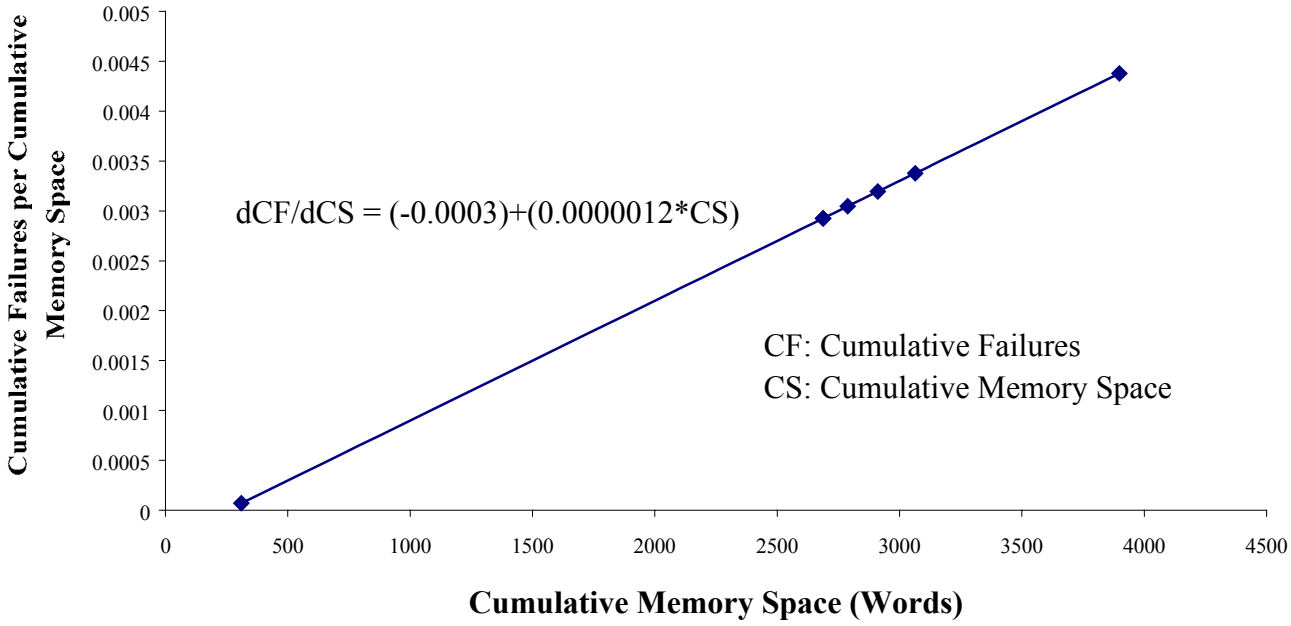

Figure 4. Rate of Change of Failures with Issues

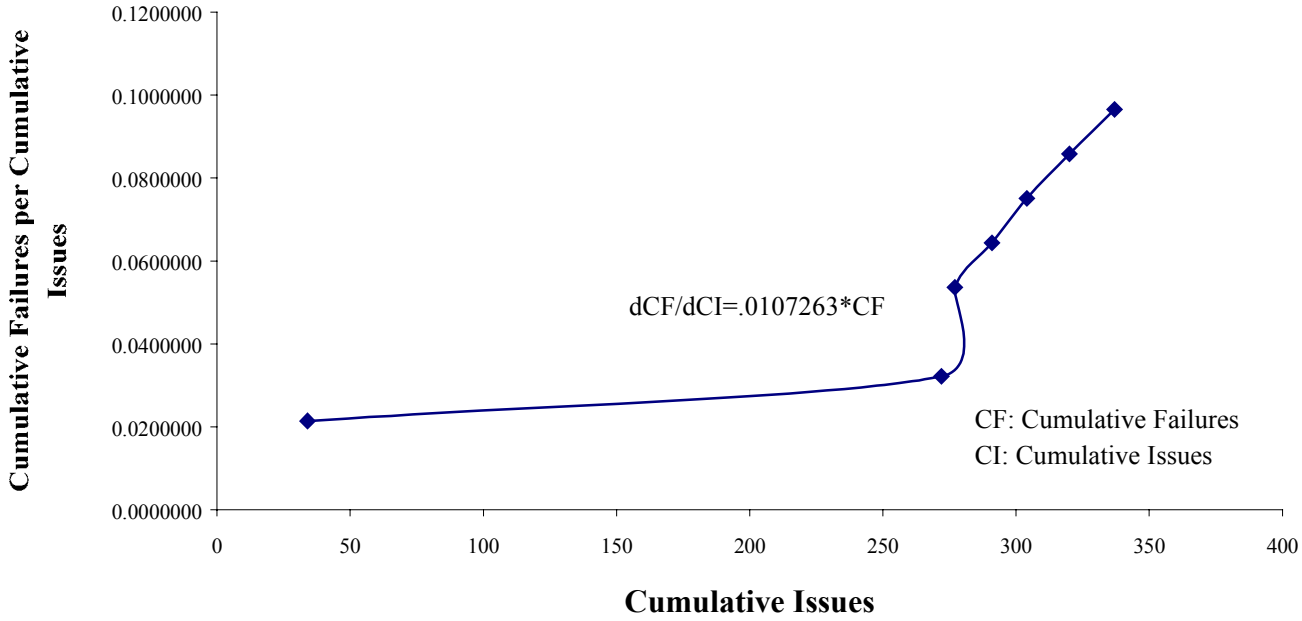

Figure 5. Memory Space vs. Failures

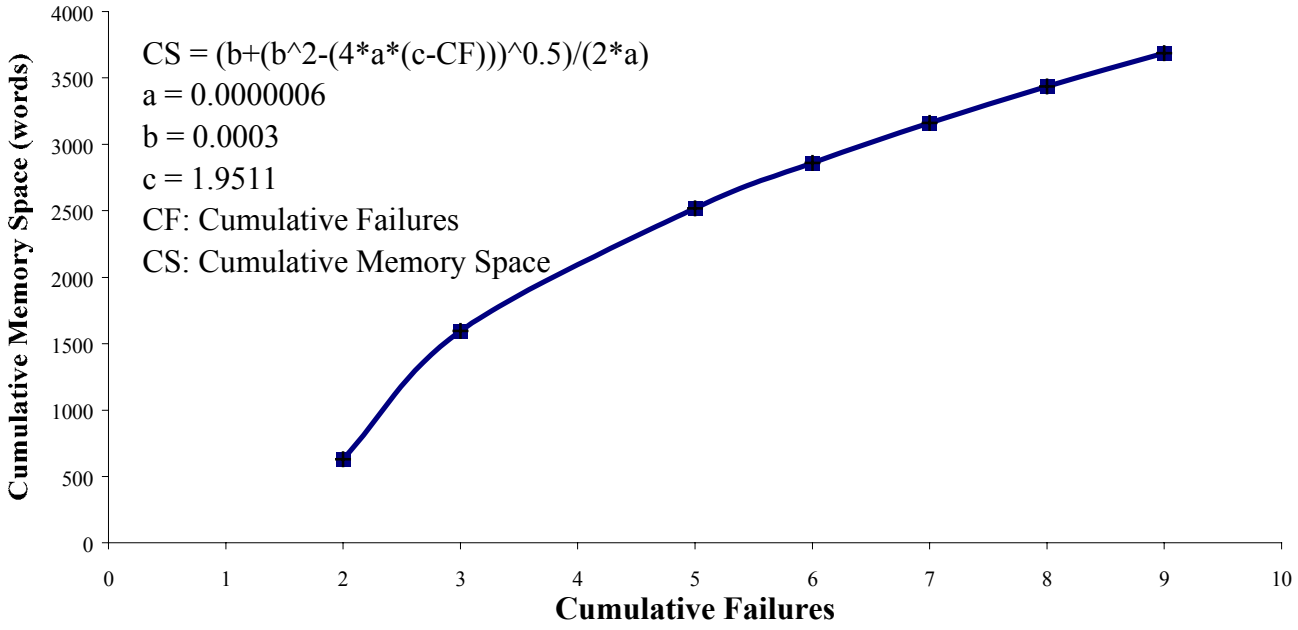

Figure 6. Issues versus Failures

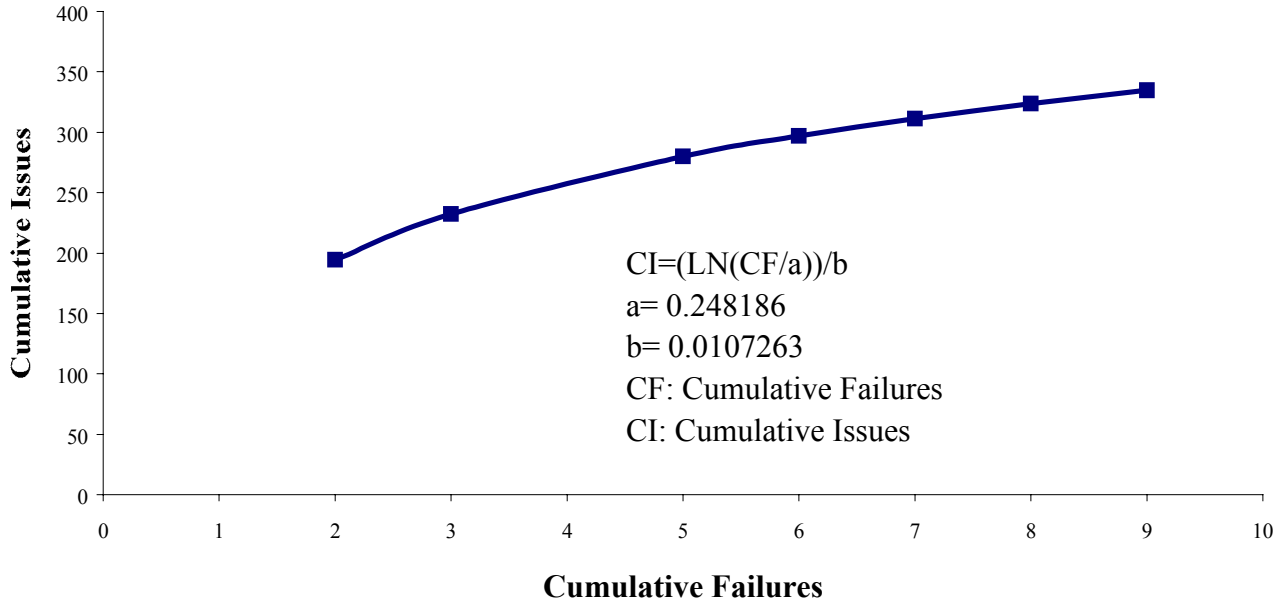

Figure 7 Mean Time To Failure Across Releases

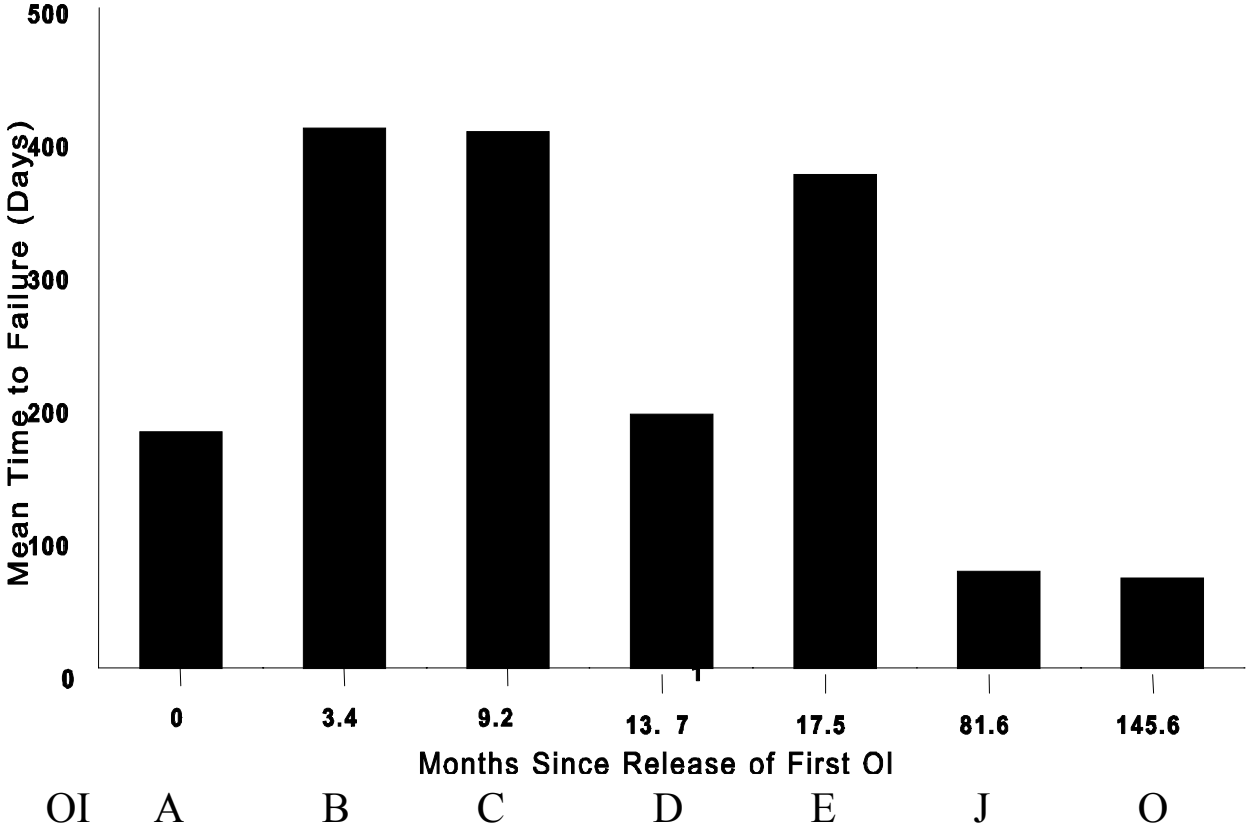

Figure 8. Total Failures per KLOC Across Releases

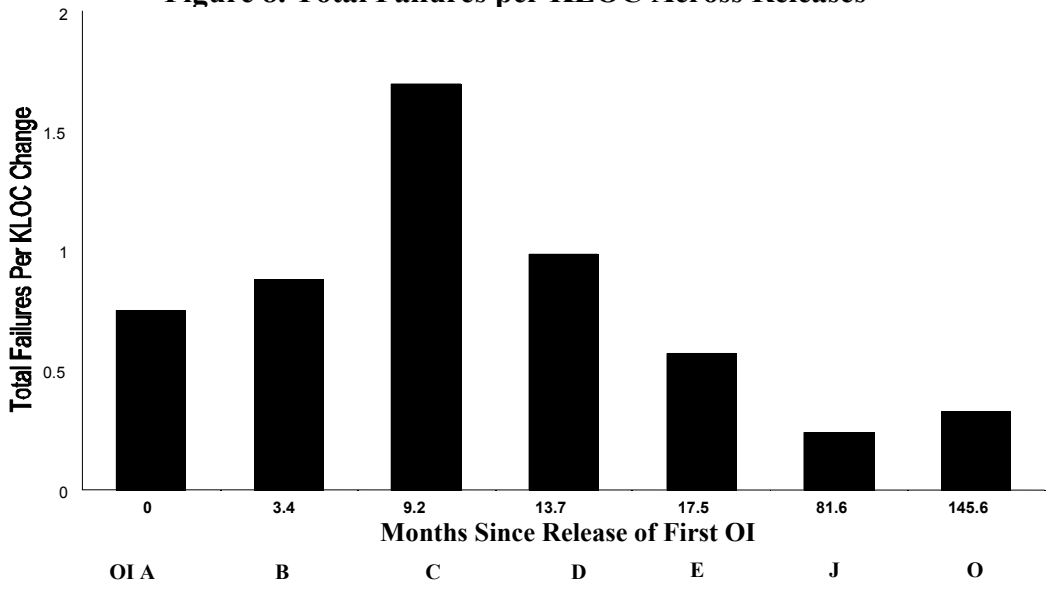

Figure 9. OIO Failure Rate

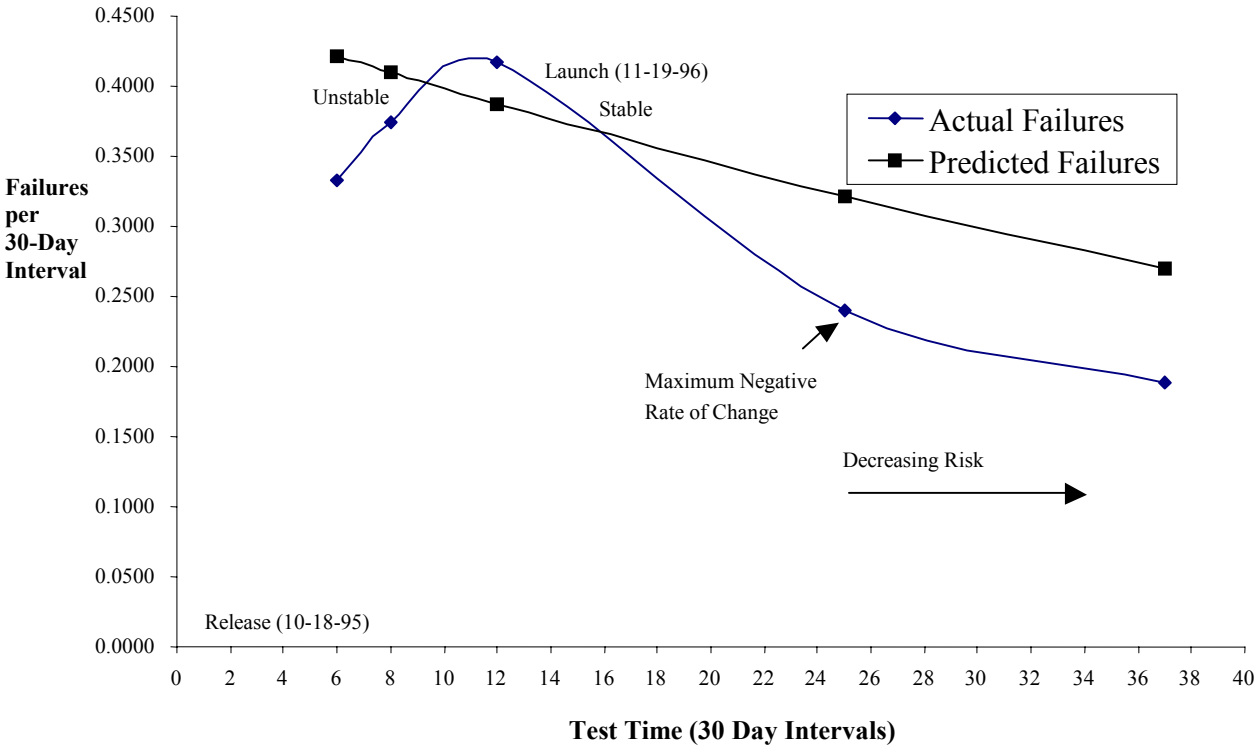

Figure 10. Predicted Number of Remaining Faults

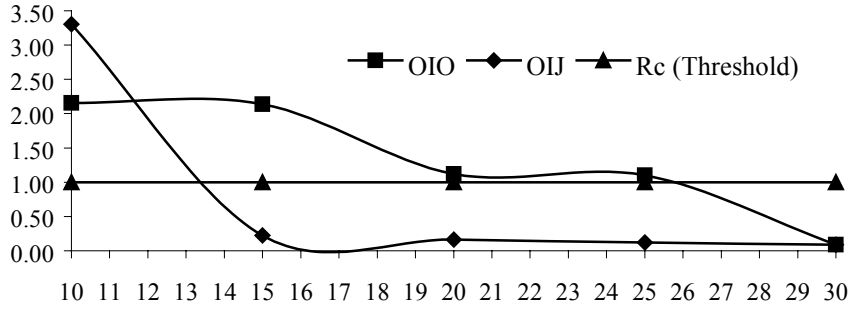

Figure 11. Predicted Proportion of Remaining Faults

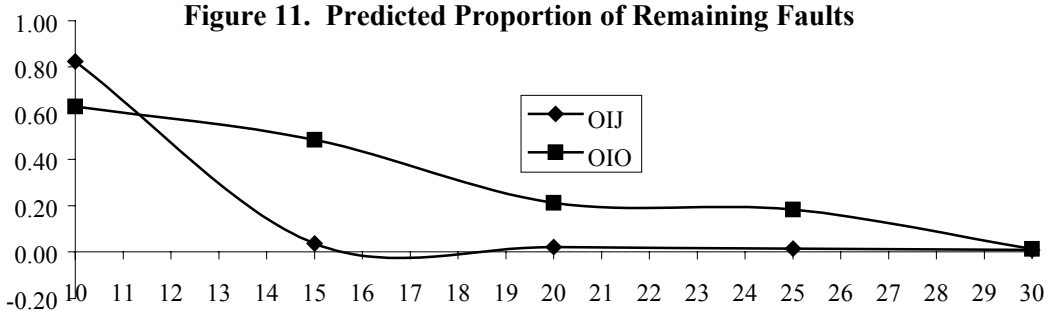

Figure 12. Predicted Time to Correct Faults